Vergleich maschineller Lernalgorithmen in Anwendung auf die 3D-Gestenerkennung per Leap Motion

Simon Fischer
Matrikelnummer: 244358

BACHELORARBEIT

zur Erlangung des Grades
Bachelor of Science
im Studiengang OnlineMedien
an der Fakultät Digitale Medien

Erstbetreuer: Prof. Dr. Matthias Wölfel
Zweitbetreuer: Prof. Dr. Dirk Eisenbiegler

Furtwangen, den 28. Februar 2017

Eidesstattliche Erklärung

Ich erkläre eidesstattlich, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen nicht benutzt und die den benutzten Quellen entnommenen Stellen als solche gekennzeichnet habe. Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegt.

Furtwangen, den 28. Februar 2017

Simon Fischer

Abstract

Die Gestensteuerung ermöglicht eine intuitive sowie natürliche Interaktion zwischen Mensch und Maschine. Um die Bewegungen des Nutzers im dreidimensionalen Raum erkennen und klassifizieren zu können, sind Sensoren nötig, deren Signale es gilt zu verarbeiten. Diese Verarbeitung geschieht mittlerweile überwiegend durch Algorithmen des maschinellen Lernens, welche mit Hilfe von eingelernten Daten selbstständig Vorhersagen über neue Eingabesignale treffen können.

In dieser Arbeit werden die Eigenschaften und die damit zusammenhängende Erkennungsgenauigkeit solcher Algorithmen untersucht und verglichen. Dabei sollen die *Support Vector Machine* und das *neuronale Netz* berücksichtigt werden, welche mit Hilfe der Open Source Bibliotheken *TensorFlow* und *scikit-learn* implementiert werden. Als Untersuchungsgegenstand dienen die Positionskoordinaten von insgesamt 350 dynamischen Gesten. Diese wurden mit dem Tiefenensor *Leap Motion* mittels einer eigens entwickelten Software aufgezeichnet und entsprechend vorverarbeitet. Auf diese Software, sowie den Prozess der Datensammlung, der Vorverarbeitung und letztendlich der Klassifizierung wird in der Arbeit ausführlicher eingegangen.

Die durchgeführten Tests zeigen, dass beide Algorithmen in der Lage sind eine Erkennungsgenauigkeit von bis zu 93,40% zu erreichen. Die Support Vector Machine schneidet durchschnittlich mit 87,15%, gegenüber dem neuronalen Netz mit 80,47%, besser ab.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Kapitel 1

Einleitung

Die Art und Weise wie wir Menschen heutzutage mit einer Maschine interagieren können ist vielfältig. Die Touchsteuerung ist seit dem Durchbruch des Smartphones kaum noch wegzudenken und auch die Sprachsteuerung findet bei immer mehr Geräten Gebrauch. Eine weitere Form der Interaktion ist die Gestensteuerung, die momentan vor allem im Spiele- und Entertainmentbereich ihren Einsatz findet. Weitere Einsatzszenarien, wie im Operationssaal, in der Krankengymnastik oder im Smart Home sind ebenfalls denkbar (vgl. Wölfel 2013). Grundlegend hierfür sind Sensoren, wie beispielsweise die *Microsoft Kinect* (siehe Han u. a. 2013) oder die *Leap Motion*[1]. Diese erkennen den Nutzer durch Kameras und zeichnen dessen Bewegungen im dreidimensionalen Raum auf. Eine andere Art von Sensoren sind solche, die der Nutzer am Körper tragen muss. In beiden Fällen gilt es die anfallenden Signale entsprechend zu verarbeiten und ihnen eine Bedeutung zuzuordnen.

Ein großer Vorteil der Gestensteuerung gegenüber den noch klassischen Eingabegeräten Maus und Tastatur besteht in der Natürlichkeit der Interaktion. Schließlich werden Gesten täglich, oft auch unbewusst, zur zwischenmenschlichen Kommunikation verwendet. Zudem ermöglicht die kamerabasierte Sensorik eine intuitivere Steuerung, indem das gewünschte Objekt direkt im dreidimensionalen Raum manipuliert werden kann. Die Interaktion mit einem weiteren Gerät wie der Maus oder gar einen Sensor am Körper tragen zu müssen, ist damit nicht mehr nötig (vgl. Garg, Aggarwal und Sofat 2009, S. 186).

Grundsätzlich können Gesten in zwei Kategorien aufgeteilt werden: in *statische Gesten*, oft auch Posen genannt, und in *dynamische Gesten* (vgl. Mitra und Acharya 2007, S. 311). Letztere unterscheiden sich insofern, dass eine beabsichtigte Bewegung über einen kurzen Zeitraum hinweg stattfindet. Diese Bewegung transportiert, wie die Pose auch, eine bestimmte Information zu einem Gegenüber, in diesem Fall einer Maschine (vgl. Wachsmuth und Fröhlich 1998, S. 198).

[1] Siehe http://www.leapmotion.com

Sobald der Sensor eine Bewegung wahrgenommen hat, muss diese einer Geste zugeordnet werden. Dafür werden mittlerweile in den meisten Ansätzen maschinelle Lernalgorithmen verwendet (vgl. D'Orazio u. a. 2016, S. 68 f.). Ein besonderer Vorteil dieser Algorithmen besteht darin, dass sie sich automatisch durch Erfahrung verbessern (vgl. Mitchell 1997, S. XV). Je mehr Daten sie verarbeiten, desto zuverlässiger können sie Entscheidungen treffen. Das bedeutet auch, dass keine hartcodierten Verfahren zur Erkennung jeder einzelnen Geste entwickelt werden müssen, sondern dem Computer wird lediglich „Wissen" antrainiert. Dadurch entsteht ein flexibles System, welches jederzeit durch weitere Gesten erweitert werden kann.

1.1 Zielsetzung

In dieser Arbeit sollen zwei maschinelle Lernalgorithmen, die *Support Vector Machine* und das *neuronale Netz*, unabhängig voneinander trainiert werden, um anschließend dynamische, nicht statische Gesten zuverlässig klassifizieren zu können. Um die Algorithmen implementieren und verwenden zu können, wird auf verschiedene Bibliotheken zurückgegriffen. Ziel ist es also nicht diese selbst zu entwickeln, sondern der Fokus liegt vielmehr auf der Untersuchung der Erkennungsgenauigkeit der einzelnen Algorithmen.

Im ersten Schritt soll ein System entwickelt werden, mit dem selbst definierte Gesten aufgezeichnet werden können. Bedingt durch den verwendeten Sensor Leap Motion, welcher nur Hand- und Fingerbewegungen erkennt, werden lediglich Handgesten berücksichtigt. Anschließend gilt es die aufgezeichneten Daten entsprechend aufzubereiten, bevor sie von der Support Vector Machine oder dem neuronalen Netz verarbeitet werden können. Zuletzt werden verschiedene Einstellungen sowie Parameter der Algorithmen und deren Auswirkungen auf die Erkennungsgenauigkeit geprüft. Für die Arbeit ergeben sich damit folgende Fragestellungen:

- Wie funktionieren die Algorithmen im Allgemeinen?
- Welche Gesten sollen aufgezeichnet werden?
- Wie müssen die Daten vorverarbeitet werden und welchen Einfluss hat das?
- Welche Gesten werden gut, welche weniger gut erkannt?
- Welche Parameter der Algorithmen haben welchen Einfluss?
- Welcher Algorithmus ist in der Lage die Gesten besser zu klassifizieren?

Ziel der Arbeit soll es nicht sein die Akzeptanz bestimmter Gesten zu untersuchen oder diese für einen konkreten Anwendungsfall zu konzipieren. Außerdem ist auch nicht vorgesehen mit dem entwickelten System eine Software oder ähnliches durch Gesten steuern zu können. Dies wäre in einem nächsten Schritt, nachdem die Algorithmen über ein bestimmtes „Wissen" verfügen, denkbar.

1.2 Aufbau der Arbeit

Insgesamt besteht die Arbeit aus acht Kapiteln. In dem folgenden Kapitel wird auf die zwei angrenzenden Forschungsgebiete, Gestenerkennung und maschinelles Lernen, eingegangen sowie der jeweilige Stand der Wissenschaft aufgezeigt. Anschließend wird in Kapitel 3 erläutert, welche Gesten aufgezeichnet und für die Arbeit verwendet wurden. Dabei wird auch auf den eingesetzten Sensor Leap Motion näher eingegangen. Kapitel 4 widmet sich dem Vorgang der Datenvorverarbeitung. In Kapitel 5 wird die Funktionsweise der zwei Algorithmen kurz beschrieben. Anschließend findet eine Vorstellung der entwickelten Software in Kapitel 6 statt. Die Ergebnisse der Arbeit werden in Kapitel 7 präsentiert. Zuletzt wird in Kapitel 8 noch mal ein zusammenfassender Überblick gegeben, woraufhin ein kurzer Ausblick zu möglichen Entwicklungen folgt.

Kapitel 2

Stand der Wissenschaft

2.1 Gestenerkennung

Die Gestenerkennung ist ein aktives Forschungsgebiet, dessen Fortschritte mit der Weiterentwicklung der dazu benötigten Sensorik einhergeht. Bereits um 1980 wurden mit Hilfe von Datenhandschuhen[1] erste Versuche zur Erkennung von Gesten unternommen (vgl. Premaratne 2014, S. 6 f.). Durch die Entwicklung von kostengünstigen Tiefensensoren, allen voran der Microsoft Kinect, wurden neue Möglichkeiten eröffnet. Denn diese Sensoren bieten, gegenüber den klassischen RGB-Kameras, welche überwiegend in den letzten Jahren in der wissenschaftlichen Forschung zum Einsatz kamen (vgl. Han u. a. 2013, S. 1325), zusätzliche Tiefeninformationen. Mit diesen Zusatzinformationen lassen sich 3D-Modelle der Umgebung rekonstruieren, womit Aufgaben der Signalverarbeitung, wie die Personensegmentierung[2], das Tracking der Person sowie die Erkennung verschiedener Körperteile vereinfacht werden (vgl. D'Orazio u. a. 2016, S. 57). Folglich sind die daraus extrahierten Daten, welche anschließend zur Klassifizierung der Gesten verwendet werden, präziser und vielfältiger. Dadurch kann auch eine größere Anzahl von Gesten differenziert und erkannt werden (vgl. Pavlovic, Sharma und Huang 1997, S. 692).

Die Erkennung von Handgesten durch kamerabasierte Sensoren stellte jedoch zu Beginn große Schwierigkeiten dar. Grund dafür ist die schwierige Differenzierbarkeit der einzelnen Finger, welche sich zudem gegenseitig verdecken können. Außerdem sind durch die Vielzahl an Gelenken die Bewegungsabläufe der Hand sehr detailliert. Infolgedessen konnte die Micorosft Kinect anfangs Handgesten nicht zuverlässig klassifizieren, was erst durch ein nachfolgendes Softwareupdate behoben werden konnte (vgl. Han u. a. 2013, S. 1329). Anders verhält sich das bei dem neueren Sensor Leap Motion, der speziell für die Erkennung von Finger- und Handbewegungen ent-

[1] Handschuhe, die mit verschiedenen Sensoren ausgestattet sind.

[2] Der Vorgang, in dem die Person von dem Hintergrund unterschieden wird.

wickelt wurde und dabei millimetergenau arbeitet (vgl. Weichert u. a. 2013, S. 6387).

Nach einer Studie von Rautaray und Agrawal (2015), in der über 250 wissenschaftliche Arbeiten untersucht wurden, wird ersichtlich, dass sowohl den dynamischen als auch den statischen Gesten gleich viel Beachtung in der wissenschaftlichen Forschung gewidmet wird.[3] Allerdings ist ebenfalls erkennbar, dass die 3D-Erkennung in Bezug auf Handgesten verhältnismäßig selten untersucht wurde. Dies hängt vermutlich mit der erst relativ neuen Sensortechnik zusammen.

2.2 Maschinelles Lernen

Maschinelles Lernen, ein Teilgebiet der Künstlichen Intelligenz, ist ebenfalls ein sehr aktives Forschungsgebiet. Durch verschiedene Errungenschaften, beispielsweise dem autonomen Fahren, gewinnt es in der Öffentlichkeit immer mehr an Bekanntheit und Popularität. Diese Disziplin beschäftigt sich dabei mit der Frage, „wie Computerprogramme entwickelt werden können, die sich automatisch durch Erfahrung verbessern“ (Mitchell 1997, S. XV). Firmen wie Google, Microsoft oder Facebook, die zuletzt ihre selbst entwickelten Algorithmen in Form von Open Source Bibliotheken veröffentlicht haben, treiben die Forschung zusätzlich voran.

Im Zusammenhang zur Gestenerkennung wird maschinelles Lernen schon seit mehreren Jahren verwendet. Bereits 1991 haben Murakami und Taguchi (1991) ein neuronales Netz eingesetzt, um 42 Gesten der japanischen Zeichensprache zu erkennen. Weitere zahlreiche wissenschaftliche Arbeiten sind in der Zwischenzeit entstanden. Neben den neuronalen Netzen werden dabei oft Hidden Markov Modelle oder Support Vector Machines zur Klassifizierung von Gesten verwendet. Allerdings hat sich über die Jahre hinweg noch kein Standard etabliert, das durch die unterschiedlichen Anwendungsfälle und deren individuelle Komplexitäten begründet werden könnte. Dennoch findet ein direkter Vergleich verschiedener Algorithmen innerhalb einer Arbeit nur selten statt (vgl. LaViola Jr. 2013, S. 10), wodurch dem Problem nur geringfügig entgegengewirkt wird.

2.3 Aufgabenstellung

Zusammenfassend lässt sich sagen, dass 3D-basierte Verfahren mittels Tiefensensoren zur Erkennung von Handgesten nur selten verwendet werden. Ebenso selten findet ein direkter Vergleich verschiedener Klassifizierungsalgorithmen statt. Auf Grund dessen wird sich diese Arbeit auf diese wenig betrachteten Inhalte der 3D-Gestenerkennung fokusieren.

[3] Siehe u. a. Binh, Shuichi und Ejima (2005), Kurakin, Zhang und Liu (2012), Lu, Tong und Chu (2016) und Marin, Dominio und Zanuttigh (2014)

Kapitel 3

Datenerhebung

Damit das neuronale Netz und die Support Vector Machine in der Lage sind die Gesten klassifizieren zu können, müssen diese vorerst mit Daten trainiert werden. Auf Grund eines fehlenden standardisierten Gestenvokabulars, sowie dem Fehlen eines geeigneten frei verfügbaren Datensatzes, mussten im ersten Schritt die Gesten definiert werden. Schließlich sollen keine willkürlichen Bewegungen des Nutzers erkannt werden. Wie dabei vorgegangen wurde und welche Gesten festgelegt wurden, wird im folgenden Abschnitt beschrieben. Letztendlich wurde mit Hilfe von sieben Personen ein Datensatz von 350 Gesten erstellt. Der Datensatz besteht dabei aus sieben verschiedenen Gesten, die je 50 mal aufgezeichnet wurden. Um die Gesten in der Länge der Ausführung zu vereinheitlichen wurde die Aufnahme auf 60 Frames[1] beschränkt, was etwa einem Zeitraum von 1-2 Sekunden entspricht.

Die Daten hätten auch nur durch den Einsatz einer Person gesammelt werden können. Der Nachteil dabei wäre jedoch, dass die Algorithmen personenspezifische Eigenschaften lernen würden, wodurch Gesten einer anderen, „unbekannten“ Person nur schwer oder gar nicht klassifiziert werden könnten.[2] Schließlich führt jede Person eine definierte Geste unterschiedlich aus. Diesem Effekt kann man, wie in Kapitel 4 beschrieben wird, zwar etwas entgegenwirken, dennoch wäre der Vorgang nicht repräsentativ. Ziel ist es die Algorithmen so generell wie möglich zu trainieren. Aus diesem Grund wurde bei der Datenerhebung auch darauf geachtet, dass jede Geste sowohl mit der rechten als auch mit der linken Hand aufgezeichnet wird.

[1] Ein Frame entspricht einem Einzelbild der Kamera, worin alle Informationen für diesen bestimmten Zeitausschnitt gespeichert sind.

[2] Das System könnte anschließend, nachdem ein „Grundwissen“ vorhanden ist, individuell durch den Nutzer trainiert werden, wodurch ein personalisiertes System entstehen würde.

0	Swipe von links nach rechts mit der vertikalen Handfläche
1	Swipe von rechts nach links mit der vertikalen Handfläche
2	Offene Handfläche zur Faust ballen
3	Zeigefinger im Uhrzeigersinn kreisen
4	Zeigefinger gegen den Uhrzeigersinn kreisen
5	Offene Handfläche zum Sensor hin bewegen
6	Offene Handfläche vom Sensor weg bewegen

Tabelle 3.1: Selbstdefinierter Gestenkatalog.

3.1 Festlegung der Gesten

Die Festlegung der Gesten beruht auf einer Umfrage aus einem Projekt der Hochschule Furtwangen unter Betreuung von Prof. Dr. Matthias Wölfel. Ziel des Projektes war es, ein gestenbasiertes Interface für Sehbehinderte zu schaffen. Dafür wurden insgesamt 82 Personen befragt, wie sie den Musikstreamingdienst Spotify per Leap Motion steuern würden. Die daraus resultierten Gesten werden in Tabelle 3.1 aufgeführt. Wie zu erkennen ist, wird lediglich eine Hand zur Ausführung verwendet.

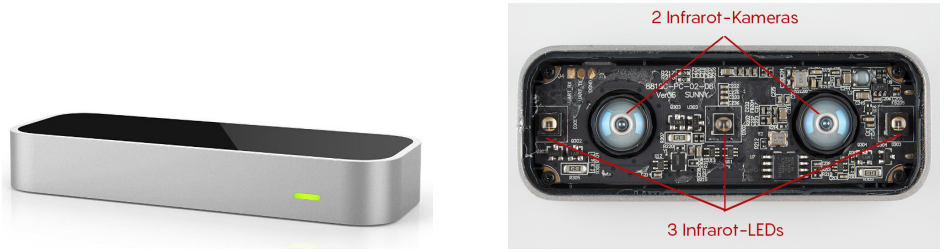
Im Folgenden soll der Sensor Leap Motion, mit dem die Gesten aufgezeichnet wurden, kurz vorgestellt werden.

3.2 Der Sensor Leap Motion

Die Leap Motion (siehe Abbildung 3.1 (a)) ist ein Sensor aus dem Jahr 2013 mit den Maßen $8,0 \times 1,2 \times 3,0$ cm. Dieser sollte möglichst auf eine gerade Oberfläche gelegt werden, wobei ein Interaktionsraum von etwa $60 \times 60 \times 60$ cm entsteht. Durch diesen relativ kleinen Raum ergibt sich auch der vorgesehene Einsatz zur Erkennung von Finger- und Handbewegungen. Wie in Abbildung 3.1 (b) zu sehen ist, verfügt die Leap Motion über drei Infrarot-LEDs und zwei Infrarot-Kameras (vgl. Weichert u. a. 2013). Über die Technik wie das Signal verarbeitet wird, wird keine Auskunft gegeben. Es heißt lediglich, dass die Bildinformationen per USB auf den Computer übertragen werden auf dem ein 3D-Modell durch die eigene Tracking-Software generiert wird. Die Positionen verdeckter Finger werden dabei anhand der verfügbaren Daten abgeleitet (vgl. Colgan 2014).

Aus diesem aufbereiteten 3D-Modell lassen sich mit Hilfe des verfügbaren SDKs[3] jeweils aus einem einzelnen Frame verschiedene Informationen,

[3]SDK steht für Software Development Kit.



(a) (b)

Abbildung 3.1: Der Leap Motion Sensor einmal von außen (a) (Quelle: Amazon 2013) und einmal von innen (b) (Quelle: SparkFun 2013).

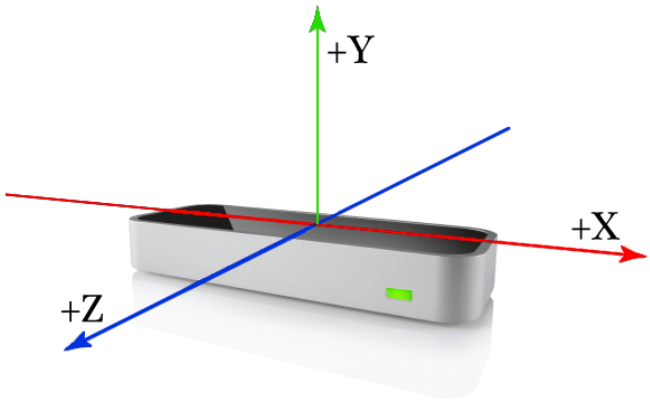


Abbildung 3.2: Das Koordinatensystem der Leap Motion (Quelle: Leap Motion 2016a).

wie die Position der einzelnen Gelenke[4], die Richtungsvektoren oder die Geschwindigkeit der Fingerspitzen abfragen (vgl. Leap Motion 2016b). Die erhaltenen Werte sind Vektoren, die die Lage der einzelnen Merkmale in Bezug auf die Leap Motion in Millimetern angeben (siehe Abbildung 3.2). Je nach Einstellungen und Rechenleistung ist die Leap Motion in der Lage bis zu 200 Frames die Sekunde aufzuzeichnen (vgl. Leap Motion 2017). Dadurch ist es möglich, trotz sehr kurzen und schnellen Gesten, detaillierte Informationen zu erhalten.

[4] englischer Begriff: Joints

Kapitel 4

Datenvorverarbeitung

Nach der Datenerhebung muss entschieden werden welche darin enthaltenen Informationen für die Repräsentation einer Geste von Bedeutung sind. Dieser Vorgang, auch bekannt als *feature extraction*, ist sehr kritisch, zumal Informationen verworfen werden. Es gilt zudem zu betrachten, wie gut die Algorithmen die extrahierten Informationen verarbeiten können, wovon letztendlich die Erkennungsgenauigkeit abhängt. Ziel ist es demnach Merkmale zu finden, die schnell berechnet werden können und zusätzlich eine hohe Differenzierbarkeit aufweisen (vgl. Bishop 2006, S. 2 f.).

Außerdem ist es sinnvoll die Daten zu normieren, um die Vergleichbarkeit derselben zu erhöhen. Für diesen Datensatz wurden personen- und ortsspezifische Informationen herausgerechnet, schließlich sollte es nicht von Bedeutung sein durch wen oder wo[1] die Geste ausgeführt wird.

4.1 Auswahl der Merkmale

In dieser Arbeit wurde die Auswahl der Merkmale vor der Datensammlung getroffen, wodurch im Vorfeld nur die relevanten Informationen gespeichert wurden. Wie bereits erwähnt liefert die Leap Motion Informationen über Position, Richtung und Geschwindigkeit einzelner Finger. In diesem Fall wurden insgesamt 25 Merkmale, bestehend aus den Gelenken, den Fingerspitzen sowie des Handmittelpunkts, betrachtet, wovon nur die jeweiligen Positionskoordinaten gespeichert wurden (siehe Abb. 4.1). Es wurden also nicht alle Informationen der Leap Motion verwertet. Ein zusammenhängendes Problem mit der Auswahl der Merkmale ist der sogenannte *Fluch der Dimensionalität*[2]. Dieser besagt, je mehr Merkmale betrachtet werden, desto mehr Beispiele werden benötigt um diese voneinander differenzieren zu können (siehe Abb. 4.2). Allerdings ist dieser Zusammenhang nicht linear, sondern exponentiell (vgl. Bishop 2006, S. 35).

[1] Solange die Geste im sichtbaren Bereich des Sensors liegt.

[2] englischer Begriff: Curse of Dimensionality

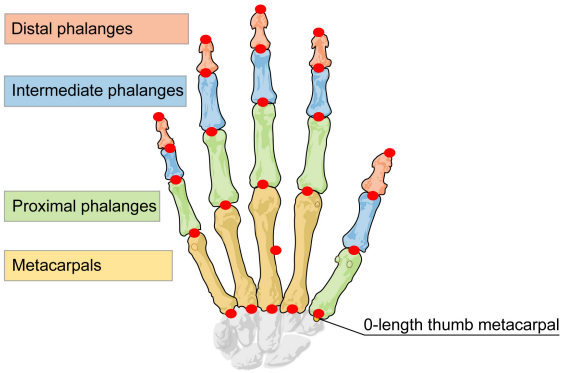


Abbildung 4.1: Darstellung des Handskeletts. Die roten Punkte verdeutlichen die für den Datensatz gespeicherten Merkmale (Quelle: Leap Motion 2016a).

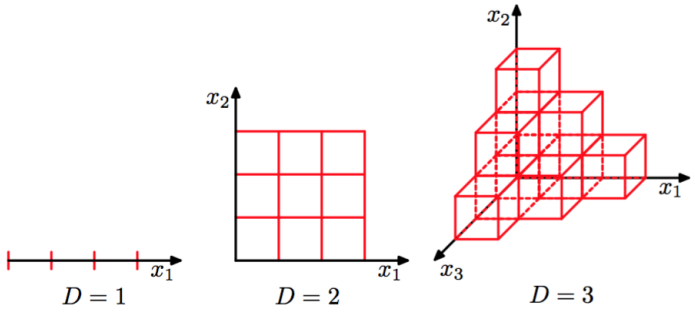


Abbildung 4.2: Illustration des Fluchs der Dimensionalität. Mit jeder weiteren Dimension steigt die Anzahl der Quadrate exponentionell an. Dementsprechend viele Trainingsdaten werden gebraucht, damit diese Zellen nicht leer bleiben (Quelle: Bishop 2006).

Durch die Begrenzung auf 60 Frames und die Auswahl der 25 Merkmale ergibt sich ein Merkmalsvektor der Länge 4500 ($25 \times 3 \times 60$). Das bedeutet, dass diese 4500 Werte eine Geste repräsentieren.

4.2 Beschneiden der aufgezeichneten Geste

Gesten bestehen aus drei Phasen: der Vorbereitung, der eigentlichen Ausführung und der anschließenden Ruhephase (vgl. Huang und Pavlovic 1995). Für die Gestenerkennung ist jedoch nur die eigentliche Ausführung relevant. Deshalb sollten die Start- und Endphase einer Geste nicht mit in die Klassifizierung einfließen. Um dies zu erreichen wurden die Aufzeichnung

vor dem Abspeichern beschnitten. Genauer gesagt wurden die Frames, die die irrelevanten Informationen beinhalten, verworfen. Damit jedoch nicht unterschiedlich lange Merkmalsvektoren entstehen, die willkürliche Gesten repräsentieren würden, mussten die gelöschten Bereiche wieder mit Nullen aufgefüllt werden.

4.3 Normierung der Daten

Ziel der Normierung ist die Vergleichbarkeit der Daten zu erhöhen. Denn die Vermutung liegt nahe, dass die Streuung der Datenpunkte einer selben Geste die Erkennungsgenauigkeit minimiert. Es muss jedoch darauf geachtet werden, dass bei der Normierung keine relevanten Informationen verloren gehen. Im schlimmsten Fall hätte dies eine unerkennbare Geste zur Folge.

4.3.1 Die Handgröße

Verschiedene Personen sollen das System trainieren und letztendlich verwenden können. Dabei sollen Informationen, wie die exakte Handgröße eines Nutzers nicht durch die Algorithmen erlernt werden. Aus diesem Grund wurden alle Hände auf eine Einheitsgröße normiert. Dazu wurde jeweils der Abstand zwischen zwei Gelenken eines Fingers berechnet und dessen Vektor normiert. Um dennoch anatomische Gegebenheiten, wie das unterschiedliche Verhältnis der Länge des Daumens zu der Länge des Zeigefingers, beizubehalten, wurde der normierte Vektor mit einem jeweiligen zuvor berechneten Skalar wieder gestreckt. Zuletzt mussten die Merkmale an die entsprechend neue Position verschoben werden (siehe Abb. 4.3).

4.3.2 Die Position

Zwischen den exakten Positionen, an denen die Gesten ausgeführt wurden, sollte ebenfalls nicht differenziert werden. Eine Hand, welche direkt über dem Sensor zur Faust geballt wird, sollte gleich behandelt werden, wie wenn diese Aktion am Rand des sichtbaren Bereichs des Sensors ausgeführt wird. Es muss allerdings darauf geachtet werden, dass bei der Verschiebung alle relevanten Informationen beibehalten werden. Beispielsweise wäre eine Verschiebung in den Mittelpunkt der Leap Motion möglich, indem der Abstand zum Handmittelpunkt normiert wird. Allerdings wird der Punkt dabei in einen Bereich von -1 bis 1 mm übertragen, wobei die restliche Hand um diesen „fixen“ Punkt rotiert. Das hat zur Folge, dass Gesten, die eine Bewegung der Handfläche beinhalten, wie es in diesem Datensatz der Fall ist, kaum bis gar nicht differenziert werden können.

Aus diesem Grund wurde der Vektor des Handmittelpunkts durch einen Skalar dividiert, wodurch die Hand in einen kleineren Bereich skaliert wird.

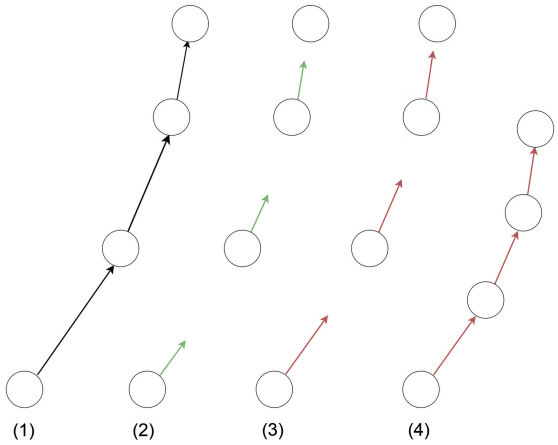


Abbildung 4.3: Veranschaulichung der Normierung eines Fingers. Im ersten Schritt wird der Abstand zweier Gelenke berechnet (1), dessen Vektor anschließend normiert (2) und mit einem Skalar gestreckt wird (3). Zuletzt müssen die Gelenke entsprechend verschoben werden (4). Der Finger wurde hier verkleinert.

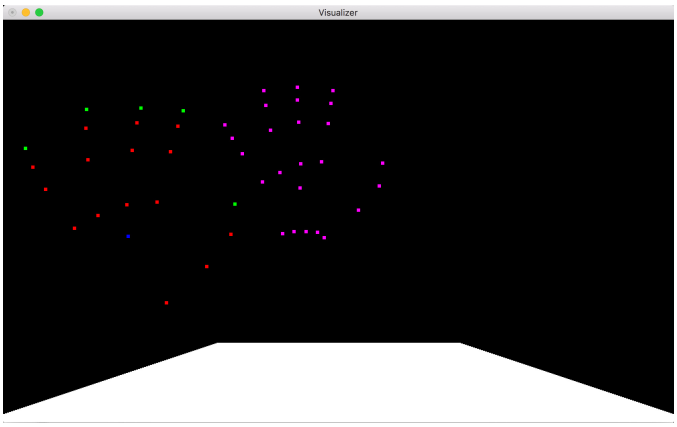


Abbildung 4.4: Visualisierung der entwickelten Software, worauf in Kapitel 6 näher eingegangen wird. Die roten und grünen Punkte repräsentieren die ausgewählten Merkmale, die von der Leap Motion erkannt werden. Die lila Punkte zeigen die selbe Hand nach der Normierung.

Ausgehend von dem verschobenen Handmittelpunkt wurden anschließend alle weiteren Positionen der Gelenke neu berechnet (siehe Abb. 4.4).

Kapitel 5

Klassifizierungsalgorithmen

Die Erkennung von Gesten wird in den meisten Fällen als Klassifizierungsproblem betrachtet, bei dem der Merkmalsvektor einer bestimmten Klasse zugeordnet werden soll. Ist die Klasse während der Trainingsphase bereits bekannt handelt es sich um das *überwachte Lernen*. Dabei muss durch einen Experten externes Wissen hinzugegeben werden, indem der Datensatz mit einem entsprechenden Label bzw. einer Klasse versehen wird (siehe Tabelle 5.1). Erst dadurch können die Algorithmen einen Zusammenhang zwischen dem Merkmalsvektor und dem Ausgangsvektor, der jeweiligen Klasse, herstellen und auf Basis dessen zukünftige, nicht beschriftete Gesten klassifizieren.

Eine andere Methode wäre das *unüberwachte Lernen*. Der Unterschied dabei ist, dass der Ausgangsvektor nicht gegeben ist. Die Algorithmen versuchen die Daten anhand von Ähnlichkeiten selbstständig zu gruppieren und strukturieren, wodurch der Merkmalsvektor keiner konkreten Geste zugeordnet werden kann.[1] Dieses Vorgehen, auch bekannt als *Clustering*, wurde für die vorliegende Arbeit nicht angewendet (vgl. Duda, Hart und Stork 2000, S. 16 f.).

5.1 Trainings- und Testphase

Im ersten Schritt müssen die Algorithmen trainiert werden, damit ein plausibler Zusammenhang zwischen dem Merkmalsvektor und der Klasse entstehen kann. Wichtig dabei ist, dass dieser Zusammenhang von den Algorithmen so gut es geht verallgemeinert wird. Die Wahrscheinlichkeit, dass die exakt selben Daten wieder eintreten, ist sehr gering (vgl. Domingos 2012, S. 80). Im darauffolgenden Schritt können Tests mit vorenthaltenen Daten durchgeführt werden, um zu überprüfen wie gut neue und unbekannte Gesten klassifiziert werden. Dementsprechend muss der gesamte Datensatz in

[1] Um ein Wissen zu generieren, benötigen die Algorithmen Annahmen, die über die verfügbaren Daten hinausgehen. Siehe dazu das „No free Lunch“ Theorem.

ID	Handmittelpunkt	Zeigefingerspitze	...	Klasse
1	[-199.35, 123.45, 30.17]	[-259.49, 143.36, -46.03]		4
2	[-41.26, 169.62, 37.72]	[11.58, 188.05, -45.03]		1
3	[50.35, 182.59, 154.33]	[44.34, 175.44, 61.90]		2
...				

Tabelle 5.1: Exemplarischer Datensatz, wobei jede Zeile einen Merkmalsvektor und somit eine Geste repräsentiert. Die Klasse, das Label, muss durch den Experten hinzugefügt werden.

zwei Sets aufgeteilt werden, einem Trainingset und einem Testset. Für den vorliegenden Fall wurden 70% der zuvor gesammelten Daten für das Training verwendet und die restlichen 30% für die Tests.

Um letztendlich die Erkennungsgenauigkeit bestimmen zu können, werden die Klassen dem Algorithmus während der Testphase vorenthalten, woraufhin die vorhergesagten Werte mit den richtigen verglichen werden. Sofern das Ergebnis nicht zufriedenstellend ist, können Änderungen an den Parametern vorgenommen werden, mit denen die Algorithmen erneut trainiert werden müssen. Daraus ergibt sich ein iterativer Prozess. Es muss jedoch darauf geachtet werden, dass es nicht zum *Overfitting* kommt, was bedeuten würde, dass die Parameter nahezu perfekt optimiert werden. Das hat zur Folge, dass den Daten zu sehr geglaubt wird und keine Generalisierung stattfindet. Die Daten innerhalb des Testsets würden zwar nahezu fehlerfrei klassifiziert werden, neue, unbekannte Daten jedoch nicht (vgl. Duda, Hart und Stork 2000, S. 12).

In den folgenden Abschnitten sollen die verwendeten maschinellen Lernalgorithmen kurz vorgestellt werden.

5.2 Künstliche Neuronale Netze

Neuronale Netze, angelehnt an das menschliche Gehirn, einem vielschichtigem Netz aus miteinander verbundenen Neuronen, sind in der Lage komplexe, nichtlineare Aufgaben zu lösen (vgl. Mitchell 1997, S.82). Die Struktur des Netzes kann in eine Eingangs- und eine Ausgangsschicht, sowie die dazwischenliegenden *Hidden Layer*[2], die in beliebiger Vielzahl auftreten können, unterteilt werden. Erstere besteht dabei aus gleich vielen Neuronen wie vorhandenen Merkmalen. Die Anzahl der Neuronen in der Ausgangsschicht ergibt sich durch die Summe der verschiedenen Klassen, wohingegen für die Hidden Layer keine Vorgabe gegeben ist (vgl. Kotsiantis 2007, S. 255). Für

[2]Die Schicht heißt „Hidden“, da die daraus entstehenden Ausgabesignale nur innerhalb des Netzes verfügbar sind (vgl. Mitchell 1997, S. 83).

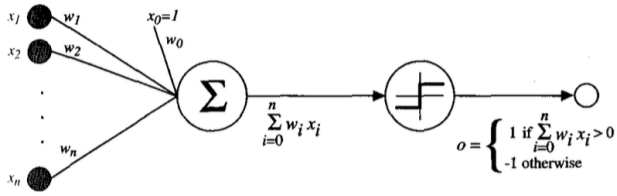


Abbildung 5.1: Ein einzelnes Neuron, dessen Aktivierungsfunktion in diesem Fall nur die Werte -1 oder 1 erzeugt (Quelle: Mitchell 1997, S. 87).

den in dieser Arbeit verwendeten Datensatz bedeutet das 4500 Neuronen in der Eingangsschicht und 7 Neuronen in der Ausgangsschicht.

Im neuronalen Netz wird das eigentliche Klassifizierungsproblem in mehrere Teilschritte zerlegt. Dazu erzeugt ein einzelnes Neuron durch relativ einfache Berechnungen aus einer Anzahl von Eingangswerten einen Ausgangswert, welcher wiederum als Eingangswert für das nächste Neuron dient (vgl. Bishop 1995, S. 117). Zuerst werden die einzelnen Eingänge mit Gewichtungen multipliziert. Daraufhin wird die Summe einer Aktivierungsfunktion[3] übergeben, welche eine negative oder positive Ausgabe, meist zwischen -1 und 1, erzeugt (siehe Abbildung 5.1). Diese Ausgabe gibt an, ob und wie stark das einzelne Neuron aktiviert wird (vgl. Duda, Hart und Stork 2000, S. 286). Wie in Abbildung 5.2 zu sehen, ergeben sich somit über das ganze Netz hinweg unterschiedlich starke Verbindungen zwischen den einzelnen Neuronen.

Dadurch dass die Klasse während der Trainingsphase bekannt ist, kann nach vollständigem Durchlaufen des Netzes eine Fehlerquote berechnet werden.[4] Dazu wird die vorhergesagte Klasse mit der richtigen Klasse verglichen. Mittels mehrerer Iterationen, in denen jede Geste erneut das Netz passiert, werden die Gewichtungen entsprechend angepasst, um diese Fehler zu minimieren (vgl. Bishop 2006, S. 241). Das neuronale Netz lernt somit selbstständig durch Erfahrung.

5.3 Support Vector Machine

Die Support Vector Machine ist ein statistisches Verfahren, mit dem die zu klassifizierenden Objekte in einen höherdimensionalen Raum übertragen und dort durch eine Hyperebene getrennt werden. Diese Ebene, auch *decision boundary* genannt, wird als optimal bezeichnet, wenn die Abstände zu den

[3] Eine häufig verwendete Aktivierungsfunktion ist die Sigmoidfunktion.

[4] Hierfür wird häufig der „Backpropagation“-Algorithmus verwendet (vgl. Kotsiantis 2007, S. 255).

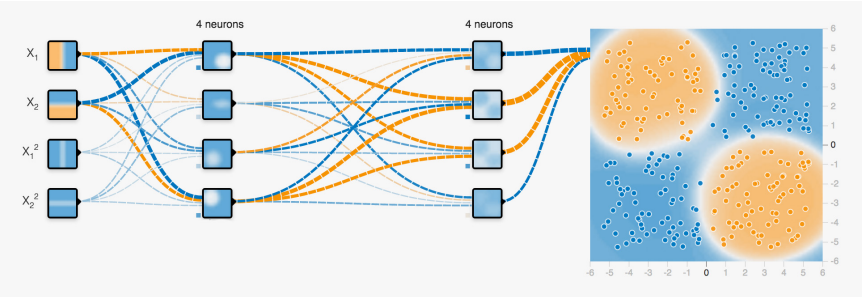


Abbildung 5.2: Ein beispielhaftes neuronales Netz mit vier Eingangsneuronen, zwei Hidden Layer und zwei Output-Neuronen, die in Form der zwei Klassen (blau und orange) dargestellt werden. Die Stärke der Linien zwischen den Neuronen verdeutlicht die unterschiedlich gewichteten Verbindungen derselben (Quelle: Smilkov und Carter 2016).

jeweiligen Klassen maximal ist. Denn dadurch kommt es zu einer Generalisierung des Algorithmus, wodurch zukünftige Daten zuverlässig eingeordnet werden können (vgl. Cortes und Vapnik 1995).

Mit Hilfe der am nächsten gelegenen Objekte, den sogenannten Stützvektoren, wird die Hyperebene aufgespannt (siehe Abb. 5.3). Folglich ist es nicht notwendig alle Punkte, besonders solche die weit weg voneinander liegen, zu berücksichtigen. Aus diesem Grund eignen sich Support Vector Machines vor allem für Anwendungsfälle, bei denen die Anzahl der Merkmale in Bezug auf den verfügbaren Datensatz sehr groß ist (vgl. Kotsiantis 2007).

Um die Klassifizierung von nicht linear trennbaren Objekten zu ermöglichen, werden die Daten in einen höherdimensionalen Raum übertragen. In diesem können die Klassen anschließend wieder linear getrennt werden (siehe Abbildung 5.4). Diese Transformation geschieht mit Hilfe eines sogenannten *Kernels*, der die Objekte in einen *Merkmalsraum* überträgt. Nachdem die Hyperebene während der Trainingsphase eingespannt wurde, überträgt der Kernel die neuen, unbekannten Daten in den selben Merkmalsraum, in dem die Klassifizierung stattfindet (vgl. Burges 1998; Cristianini und Shawe-Taylor 2000).

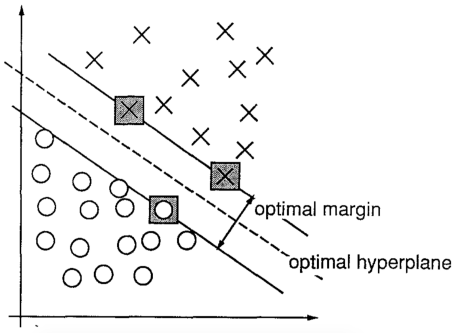


Abbildung 5.3: Lineare Support Vector Machine. Die grau markierten Objekte verdeutlichen die Stützvektoren (Quelle: Cortes und Vapnik 1995, S. 275).

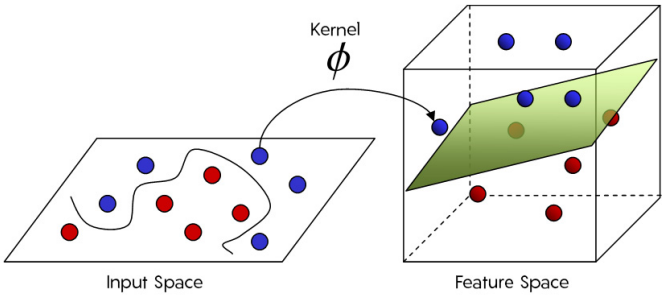


Abbildung 5.4: Übertragung der Merkmale durch einen Kernel in einen höherdimensionalen Merkmalsraum (Quelle: karenu 2012).

Kapitel 6

Vorstellung der entwickelten Software

Das Programm, mit dem die Daten gesammelt, vorverarbeitet und klassifiziert wurden, wurde mit Python[1] entwickelt. Allerdings ist zu erwähnen, dass keine zusammenhängende Software vorliegt, welche über eine grafische Oberfläche bedient werden kann. Es sind lediglich einzelne Python Module, die über die Kommandozeile ausgeführt werden müssen. Grundsätzlich gibt es drei voneinander getrennte Blöcke, die unterschiedliche Aufgaben erfüllen, wozu weitere (Sub-)Module importiert werden. Diese Abhängigkeiten zueinander werden in Abbildung 6.1 dargestellt.

Das erste Modul *leapController.py* dient dazu, die Leap Motion auf Basis dessen Bibliothek anzusteuern (siehe Abb. 6.2) und die empfangenen Daten

[1] https://www.python.org/

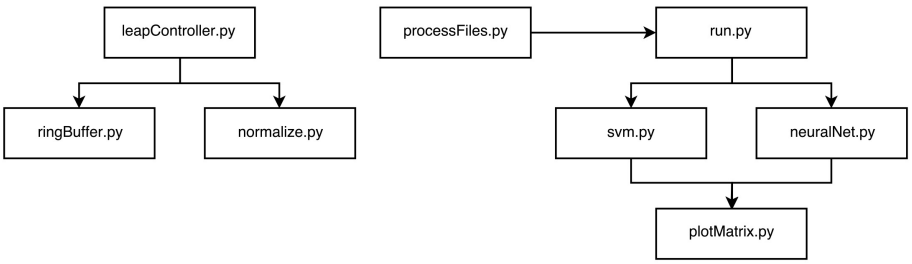


Abbildung 6.1: Abbildung der einzelnen Software-Module und deren Zusammenhänge. Mit dem ersten Modul *leapController.py* können die Gesten gesammelt und bearbeitet werden. Diese Daten werden mit Hilfe von *processFiles.py* jeweils in einen Trainings- und Testdatensatz aufgeteilt. Die anschließende Klassifizierung sowie die Tests werden mit *run.py* ausgeführt.

auf dem Bildschirm zu visualisieren. Für die Visualisierung wurde OpenGL[2], genauer gesagt PyOpenGL[3] verwendet. Damit ist es möglich die Vektoren in einem virtuellen, dreidimensionalen Raum abzubilden. Neben den Daten, die in Echtzeit empfangen werden, wird zusätzlich ein Zwischenspeicher visualisiert (siehe Abb. 4.4). Dieser beinhaltet die ausgewählten Sensordaten der letzten 60 Frames. Aus diesem Zwischenspeicher, auch Ringbuffer genannt, können die Gesten anschließend vorverarbeitet und abgespeichert werden. Dazu nimmt das OpenGL-Fenster Tastaturbefehle entgegen, wodurch eine dazugehörige Funktion aufgerufen werden kann (siehe. Abb. 6.3). Je nach Aktion ist eine weitere Interaktion mit der Kommandozeile nötig. Beispielsweise wird beim Abspeichern des Datensatzes eine ID, welche für den Dateinamen benötigt wird, abgefragt, woraufhin der Nutzer ebenfalls die jeweilige Klasse (0–7) definieren muss. Zuletzt werden die Daten im CSV-Format sowohl unbearbeitet, so wie sie aufgezeichnet wurden, als auch bearbeitet als getrennte Dateien abgespeichert.

Das zweite Modul *processFiles.py* generiert aus den einzeln abgespeicherten CSV-Dateien einen großen Datensatz und daraus wiederum jeweils einen Datensatz für die Trainings- und Testphase. Dazu muss lediglich der Ordner, in dem sich die Dateien befinden sowie der Pfad des generierten Datensatzes angegeben werden. Außerdem ist es möglich die Größe des Trainingsets durch einen prozentualen Wert normiert zwischen 0 und 1 zu definieren (siehe Abb. 6.4). Die Daten werden zufällig sortiert, um weitere Trainingsdurchläufe mit unterschiedlichen Datensätzen zu ermöglichen. Dabei wird jedoch überprüft, ob wenigstens ein Element jeder Klasse in beiden Datensätzen vorhanden ist.

Die beiden maschinellen Lernalgorithmen wurden mit Hilfe von zwei frei zugänglichen Bibliotheken implementiert. Für das neuronale Netz wurde Google's *TensorFlow*[4] und für die Support Vector Machine *scikit-learn*[5] verwendet. Der Grund für die Wahl zweier Bibliotheken ist, dass zuerst auf TensorFlow, aufgrund Googles Reputation, gesetzt wurde, diese jedoch die Support Vector Machines (noch) nicht im vollen Umfang unterstützt. Grundsätzlich unterscheiden sich beide Bibliotheken im Grundgedanken. scikit-learn bietet dem Anwender bereits gebrauchsfertige Algorithmen, die direkt eingesetzt werden können. TensorFlow hingegen ist eher als Baukasten zu verstehen mit dem sich maschinelle Lernalgorithmen, mit dem Fokus auf neuronale Netze, modellieren lassen. Dennoch beinhaltet TensorFlow ein vereinfachtes Interface, namens *TF Learn*[6], welches an scikit-learn angelehnt ist. Folglich ist es möglich mit beiden Bibliotheken durch wenige Zeilen Code

[2] https://www.opengl.org/

[3] http://pyopengl.sourceforge.net/

[4] https://www.tensorflow.org/; siehe auch Abadi u. a. 2015

[5] http://scikit-learn.org/; siehe auch Pedregosa u. a. 2011

[6] https://github.com/tensorflow/tensorflow/tree/master/tensorflow/contrib/learn/python/learn

```
import Leap
import ringBuffer

class Visualizer():
    # Loop Function
    def draw(self):
        features = []
        # get Leap Data
        frame = controller.frame()

        for hand in frame.hands:
            hand_center = hand.palm_position.to_float_array()
            # Save x, y, z separately
            for axis in range(0, 3):
                features.append(hand_center[axis])

            for finger in hand.fingers:
                # Get each Bone
                for b in range(0, 4):
                    bone = finger.bone(b).prev_joint.to_float_array()
                    for axis in range(0, 3):
                        features.append(bone[axis])

        # Add Frame to RingBuffer
        rb.append(features)

def main():
    global controller, nFrames, rb
    controller = Leap.Controller()
    nFrames = 60    # Number of Frames
    rb = ringBuffer.RingBuffer(nFrames)
    Visualizer()
```

Abbildung 6.2: Der sehr vereinfachte Code aus *leapController*.py zeigt, wie die Daten der Leap Motion abgefragt und in dem Ringbuffer abgespeichert werden.

die Algorithmen zu verwenden.

Die Algorithmen wurden jeweils getrennt in einem entsprechenden Modul, respektive in *neuralNet.py* (siehe Abb. 6.5) und *svm.py* (siehe Abb. 6.6) implementiert. Wichtig dabei ist zu erwähnen, dass sämtliche Bezugswerte, wie die Anzahl an Merkmalen und Klassen, die zur Initialisierung nötig sind dynamisch generiert werden. Dadurch gibt es keine Beschränkung auf den in der Arbeit verwendeten Merkmalsvektor der Länge 4500, sondern es lassen sich auch andere Datensätze problemlos verarbeiten. Der Nutzer muss lediglich ein Objekt der Klasse mit den jeweiligen Datensätzen erzeugen. Dieser Vorgang wird durch *run.py* vereinfacht, womit die Algorithmen mit unterschiedlichen Parametern, wie der zu verwendende Kernel oder die Struktur

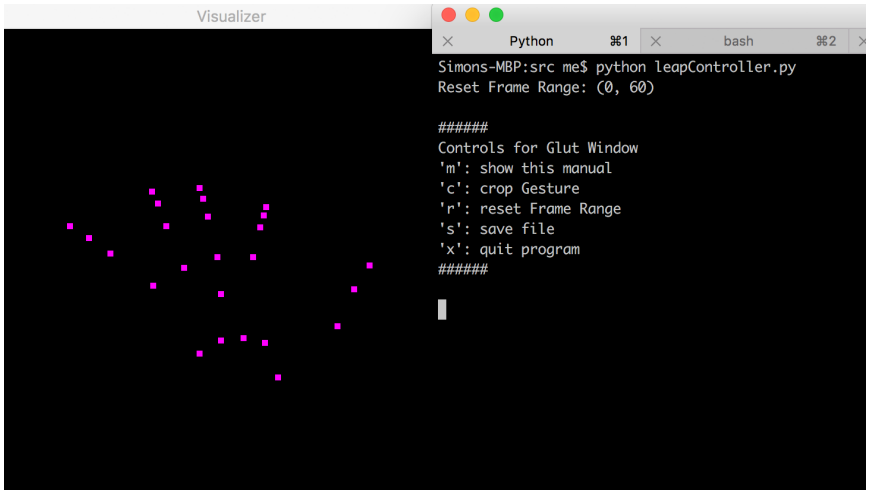


Abbildung 6.3: Veranschaulichung der Softwaresteuerung. Links ist das OpenGL-Fenster, welches Tastaturbefehle entegen nimmt, zu sehen, während rechts die Befehle in der Konsole aufgelistet werden. Bei einigen Aktionen ist eine anschließende Interaktion mit der Konsole nötig.

```
1  def mergeCSV(srcDir, output):
2      # walk through directory, read each csv and write one big csv
3      ...
4
5      return output
6
7  def splitFiles(file,
8                 train_file='../data/training_set.csv',
9                 test_file='../data/test_set.csv',
10                train_size=0.5):
11     # read csv, save as list, shuffle data, split list
12     ...
13
14 def main(dir="../data/",
15          dataset='../data/dataset.csv',
16          train_size=0.7):
17
18     splitFiles(mergeCSV(dir, dataset), train_size)
19     print("Files successfully generated")
```

Abbildung 6.4: Die Funktionen aus *processFiles.py*. In der main-Funktion wird in Zeile 18 die Funktion splitFiles mit dem von mergeCSV zuvor generierten Datensatz, sowie einem Wert für die Größe des Trainingsets aufgerufen.

```
import tensorflow as tf

class NN():
    def __init__(self,
                 trainingFile="example.csv",
                 testFile="example.csv"):
        # Load Datasets
        ...

    def main(self, layers=[450]):
        # Neural Network
        classifier = tf.contrib.learn.DNNClassifier(
                        hidden_units=layers,
                        ... # More Parameters
                        )
        # Train classifier
        classifier.fit(x=training_set.data,
                       y=training_set.target)

        # Evaluate Accuracy
        accuracy_score = classifier.evaluate(
                            x=test_set.data,
                            y=test_set.target)["accuracy"]

        return accuracy_score
```

Abbildung 6.5: Vereinfachte Darstellung der Implementierung eines neuronalen Netztes mit Hilfe von TensorFlow.

des neuronalen Netzes aufgerufen werden können (siehe Abb. 6.7).

Des Weiteren wurde Numpy[7] zu verschiedenen Vektor-Berechnungen verwendet. Die Konfusionsmatrix sowie die Lernkurve der Support Vector Machine wurden ebenfalls mit Hilfe von scikit-learn berechnet und anschließend mit Matplotlib[8] erzeugt.[9] Diese Funktionen befinden sich in *plotMatrix.py*

[7] http://www.numpy.org/

[8] http://matplotlib.org/

[9] siehe scikit-learn developers 2016a; scikit-learn developers 2016b

```
from sklearn import svm

class SVM():
    def __init__(self,
                 trainingFile="example.csv",
                 testFile="example.csv"):
        # Load datasets
        ...

    def main(self, kernel="poly"):
        # Support Vector Machine
        classifier = svm.SVC(kernel=kernel,
                             degree=2,
                             ... # More Parameters
                             )
        # Train classifier
        classifier.fit(training_set_data, training_set_target)

        # Evaluate Accuracy
        accuracy_score = classifier.score(test_set_data,
                                          test_set_target)

        return accuracy_score
```

Abbildung 6.6: Vereinfachte Darstellung der Implementierung einer Support Vector Machine mit Hilfe von scikit-learn.

```
import neuralNet
import svm

trainingFile = "../data/training_set.csv"
testFile = "../data/test_set.csv"

estimator1 = neuralNet.NN(trainingFile=trainingFile, testFile=testFile)
estimator2 = svm.SVM(trainingFile=trainingFile, testFile=testFile)

estimator1.main(layers=[450, 200])
estimator2.main(kernel="poly")
```

Abbildung 6.7: Aufruf der zwei Algorithmen per *run.py*. Verschiedene Tests, in Form von Schleifen oder unterschiedlichen Parametern, lassen sich so einfach ausführen.

Kapitel 7

Ergebnisse und Auswertung

Die Ergebnisse werden in Form der Konfusionsmatrix dargestellt (siehe Abb. 7.1). Dabei bildet die x-Achse die durch den Algorithmus vorhergesagte Klasse ab, während die richtige Klasse auf die y-Achse übertragen wird. Durch diese Darstellung lässt sich nachvollziehen, wie sich die Erkennungsgenauigkeit zusammensetzt und welche Gesten dabei falsch klassifiziert wurden.

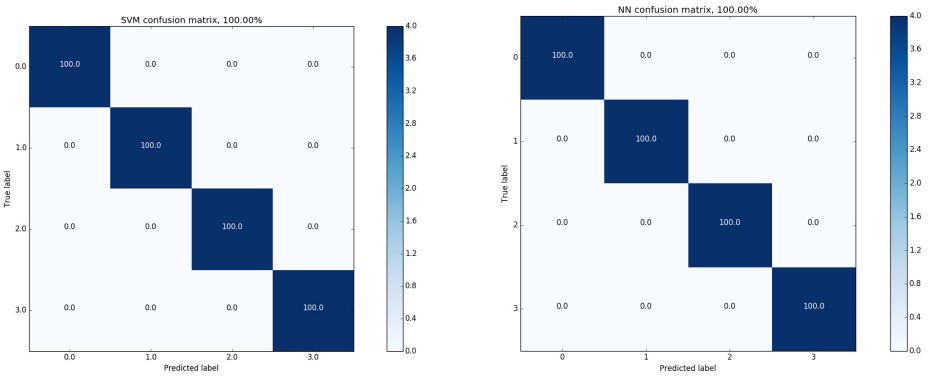
Noch anzumerken ist, dass die Struktur eines neuronalen Netzes im Folgenden, sofern nicht anders erwähnt, lediglich durch die Hidden Layer beschrieben wird. Schließlich bleiben die Eingangsschicht mit 4500 Neuronen und die Ausgangsschicht mit 7 Neuronen immer gleich.

7.1 Erste Tests mit Posen

In den ersten Durchläufen, um die Abläufe der Software zu testen, wurde mit statischen Posen gearbeitet. Hierfür wurden nur sechs Merkmale mit den jeweiligen 3D-Koordinaten über zehn Frames hinweg aufgezeichnet, wodurch sich ein deutlich geringerer Merkmalsvektor der Länge 180 ergibt ($6 \times 3 \times 10$). Wie in Abbildung 7.1 zu sehen ist, konnten die Support Vector Machine sowie das Neuronale Netz die Testdaten fehlerfrei klassifizieren. Das bedeutet jedoch nicht, dass zukünftige Posen genauso zuverlässig erkannt werden können. Denn der verwendete Datensatz, der nur die Gesten einer Person beinhaltet, ist sowohl mit 48 Aufzeichnungen als auch mit nur vier verschiedenen Posen sehr klein. Dennoch ist das Ergebnis aussichtsreich.

7.2 Vergleich der maschinellen Lernalgorithmen

Sowohl die Support Vector Machine als auch das Neuronale Netz konnten anhand eines Datensatzes eine Erkennungsgenauigkeit von 93,40% erzielen. In Abbildung 7.2 ist die jeweils dazugehörige Auswertung zu sehen. Es lässt sich feststellen, dass die Gesten von den Algorithmen unterschiedlich gut



(a) (b)

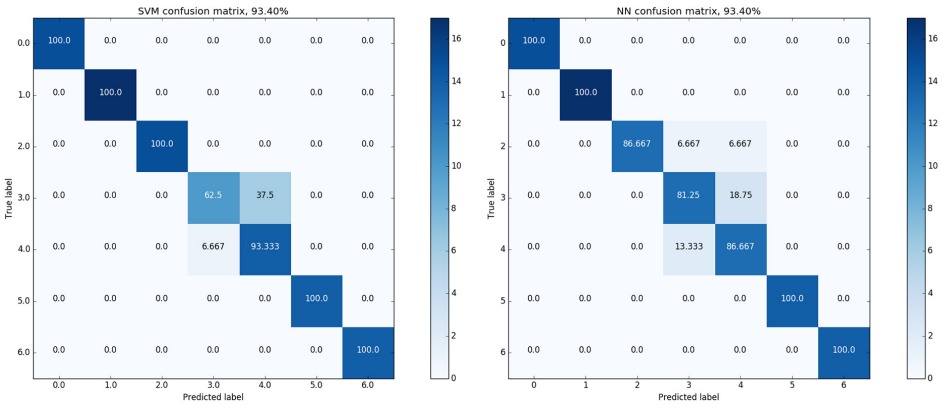
Abbildung 7.1: Konfusionsmatrix der Klassifizierungsergebnisse von 16 statischen Posen. Die Support Vector Machine (a) verwendet dabei einen linearen Kernel und das neuronale Netz (b) eine gesamte Netzstruktur von 180-20-10-20-4 Neuronen.

erkannt wurden. Während die Support Vector Machine (Abb. 7.2 a) Geste 3 nur zu 62,5% erkannte, wurde selbige vom neuronalen Netz (Abb. 7.2 b) zu 81,25% erkannt. Die Gesten 2 und 4 wurden wiederum von der Support Vector Machine besser erkannt.

In der Konfusionsmatrix kann man ebenfalls sehr gut erkennen, dass die Gesten 3 und 4 relativ häufig miteinander verwechselt wurden. Allerdings ist dabei zu berücksichtigen, dass diese Gesten auch sehr ähnlich sind: einmal den Zeigefinger *im* und einmal *gegen* den Uhrzeigersinn kreisen (siehe Tabelle 3.1). Daraus lässt sich schließen, dass Gesten, bei denen sich die Positionskoordinaten vieler Merkmale ändern, fehlerfrei klassifiziert werden konnten. Feinere Gesten, bei denen nur geringfügige Bewegungsänderungen vorgenommen werden, konnten dagegen zwar gut, aber schwerer differenziert werden.

In Abbildung 7.3 ist die Lernkurve der Support Vector Machine zu sehen. Die Kurve zeigt an, dass die Erkennungsgenauigkeit mit der Menge an Daten zunimmt. Es lässt sich vermuten, dass diese mit zusätzlichen Daten noch weiter ansteigen würde.

Die Lernkurven des neuronalen Netzes werden in Abbildung 7.4 dargestellt, wobei die Anzahl der Iterationen, in denen das Netz die Daten verarbeitet, auf der x-Achse abgebildet wird. Die Erkennungsgenauigkeit (Abb. 7.4 a) sowie die dazugehörige Fehlerkurve (Abb 7.4 b) steigen bzw. sinken verhältnismäßig zueinander. Jedoch haben sich bereits nach 400 Iterationen die Gewichtungen eingependelt, wodurch weitere Trainingseinheiten keine Optimierungen mit sich bringen. Neuronale Netze können folglich nicht endlos mit denselben Daten trainiert werden.



(a) (b)

Abbildung 7.2: Konfusionsmatrix der Klassifizierungsergebnisse von etwa 100 dynamischen Gesten. Die Support Vector Machine (a) verwendet ein Polynom zweiten Grades als Kernel und das neuronale Netz (b) einen Hidden Layer mit 450 Neuronen.

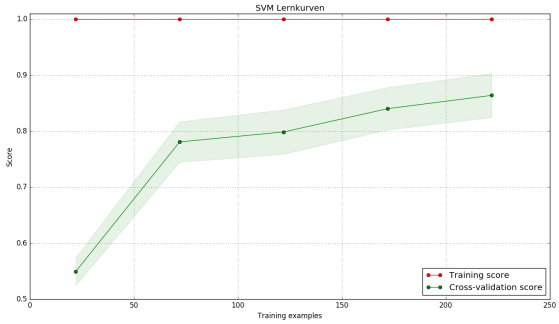
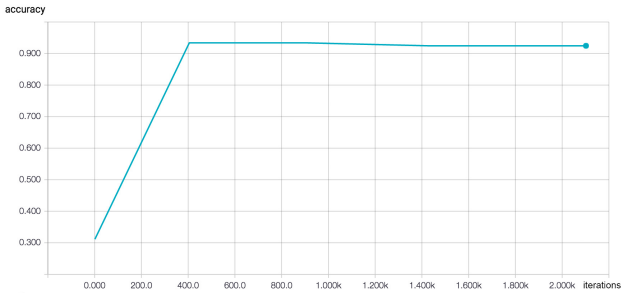


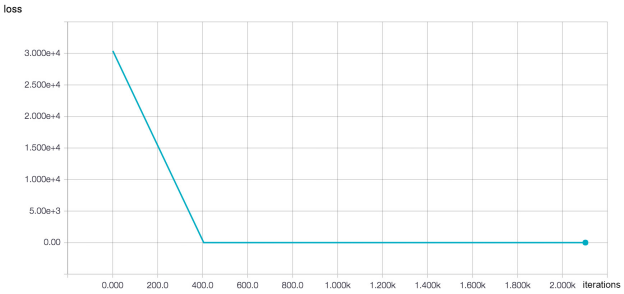
Abbildung 7.3: Lernkurve der Support Vector Machine, die mit zunehmend mehr Daten ansteigt.

7.3 Vergleich verschiedener Parameter

In diesem Abschnitt werden Ergebnisse präsentiert, die jeweils durch verschiedene Parameter erzielt wurden. Bei der Support Vector Machines wurden hauptsächlich die Kernels ausgetauscht, während beim neuronalen Netz die Netzstruktur geändert wurde. Ziel hierbei war es herauszufinden, welche Auswirkungen diese Änderungen haben und daraus resultierend die besten Einstellungen zu erreichen. Um einen Vergleich herstellen zu können wurde jeweils derselbe Trainings- und Testdatensatz verwendet. Von diesem Datensatz stammen auch die im vorherigen Abschnitt bereits gezeigten Ergebnisse.



(a)



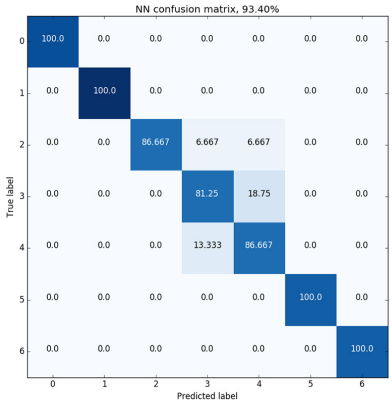
(b)

Abbildung 7.4: Die Lernkurven eines neuronalen Netzes. Die steigende Erkennungsgenauigkeit (a) sowie die sinkende Fehlerkurve (b) erreichen ihr Optimum bereits nach 400 Iterationen. Sämtliche weiteren Trainigsrunden haben keinen weiteren Einfluss auf die Klassifizierung.

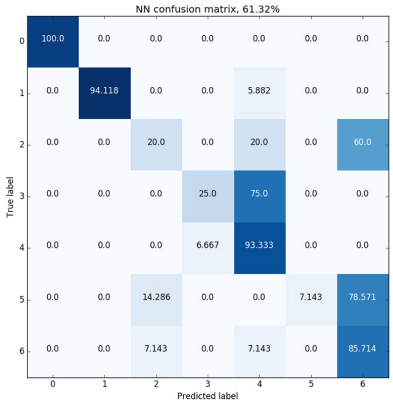
7.3.1 Neuronale Netze

In Abbildung 7.5 sind die Auswertungen fünf verschiedener neuronaler Netze aufgeführt. Das beste Ergebnis von 93,40% wurde mit einem Hidden Layer bestehend aus 450 Neuronen erreicht (Abb. 7.5 a). Die Vermutung liegt nahe, dass die Genauigkeit durch das Hinzufügen von weiteren Schichten steigt. Dem ist jedoch nicht so, denn durch einen weiteren Hidden Layer sank die Erkennungsgenauigkeit um etwas mehr als 30% (Abb. 7.5 b). Zwar stieg diese wiederum etwas mit dem dritten Hidden Layer an, dennoch ist die Erkennung im Vergleich zum einschichtigen Netz um knapp 20% schlechter (Abb. 7.5 c). Außerdem zeigen die Ergebnisse, dass die Anzahl der Neuronen weder zu klein (Abb. 7.5 d), noch zu groß (Abb. 7.5 e) sein darf.

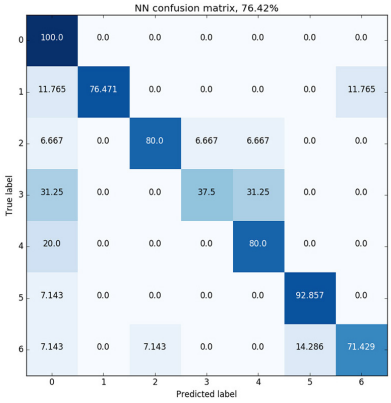
Insgesamt lässt sich sagen, dass die Struktur des neuronalen Netzes sehr kritisch ist, da die Anzahl der Neuronen sowie der Hidden Layer einen erheblichen Einfluss auf die Erkennungsgenauigkeit haben.



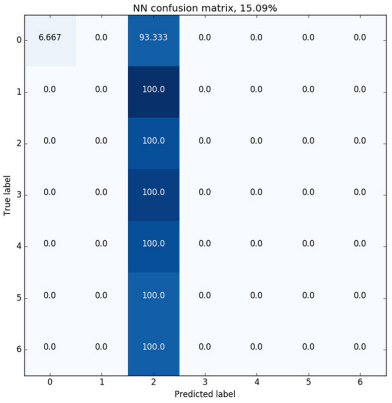
(a) 450



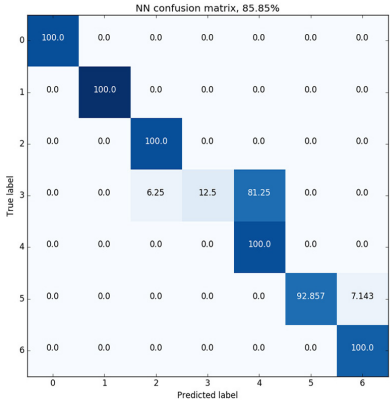
(b) 450-200



(c) 450-200-300



(d) 10



(e) 12000

Abbildung 7.5: Auswertung verschiedener Neuronaler Netze. Die Zahl unterhalb der Matrix gibt die Struktur des Netzes an. 450-200 bedeutet somit, dass das Netz aus zwei Hidden Layer mit 450 Neuronen in der ersten und 200 Neuronen in der zweiten Schicht besteht.

7.3.2 Support Vector Machines

Bei den Support Vector Machines wurden die Erkennungsgenauigkeiten unter Verwendung verschiedener Kernels ausgewertet (siehe Abbildung 7.6), wobei sich eine Spanne von etwa 80% feststellen lässt. Am besten konnten der lineare Kernel (Abb. 7.6 a) und der polynomische Kernel zweiten Grades (Abb. 7.6 b) mit jeweils 93,40% abschneiden. Bei komplexeren Funktionen, wie Polynome höheren Grades (Abb. 7.6 c & d), der radialen Basisfunktion (Abb. 7.6 e) oder der Sigmoidfuntkion (Abb. 7.6 f) nahm die Genauigkeit zunehmend ab.

Ähnlich wie bei den neuronalen Netzen ist die Wahl des richtigen Kernels für die Erkennungsgenauigkeit entscheidend.

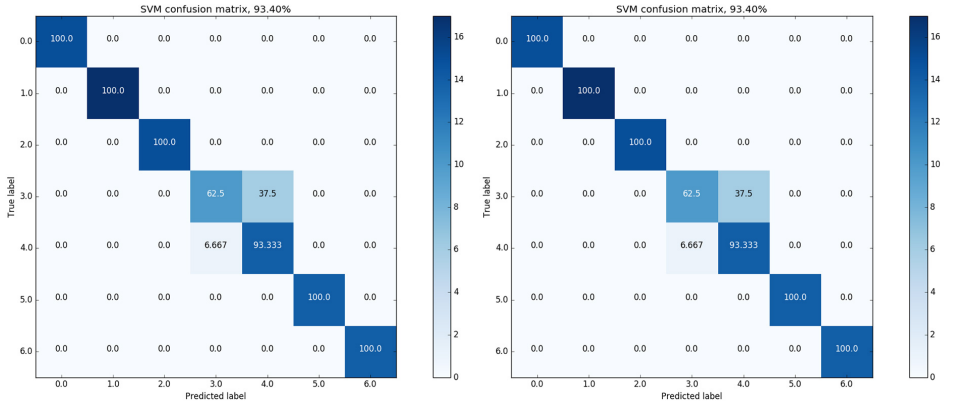
7.4 Vergleich verschiedener Datensätze

Zuletzt wurden mehrere Tests durchgeführt, bei denen die Datensätze unterschiedlich, jedoch mit dem selben Verhältnis von 70 zu 30 Prozent, aufgeteilt wurden. Die Daten selbst sind die gleichen geblieben, nur die Unterteilung in Training- und Testdatensatz unterscheidet sich. Mit jedem Datensatz-Paar wurden die Algorithmen von Neuem trainiert. Es handelt sich dabei um das selbe neuronale Netz mit 450 Neuronen sowie der selben Support Vector Machine mit dem Polynom zweiten Grades als Kernel wie aus 7.2.

Die Erkennungsgenauigkeit beider Algorithmen unterscheidet sich bei fast jedem Durchlauf. Die Qualität der Daten innerhalb des Trainingsets sind somit von großer Bedeutung. Schließlich sind sie entscheidend dafür, wie gut die Algorithmen Zusammenhänge zwischen dem Merkmalsvektor und der jeweiligen Klasse herstellen können. Die Support Vector Machine (siehe Tabelle 7.2) erreichte bei zehn eindeutigen Werten einen Durchschnitt von 87,15% und schnitt damit um knapp 7% besser ab als das neuronale Netz (siehe Tabelle 7.1). Auffällig bei beiden Algorithmen ist, dass bei allen Durchläufen das beste Ergebnis von 93,40% nicht mehr erreicht wurde.

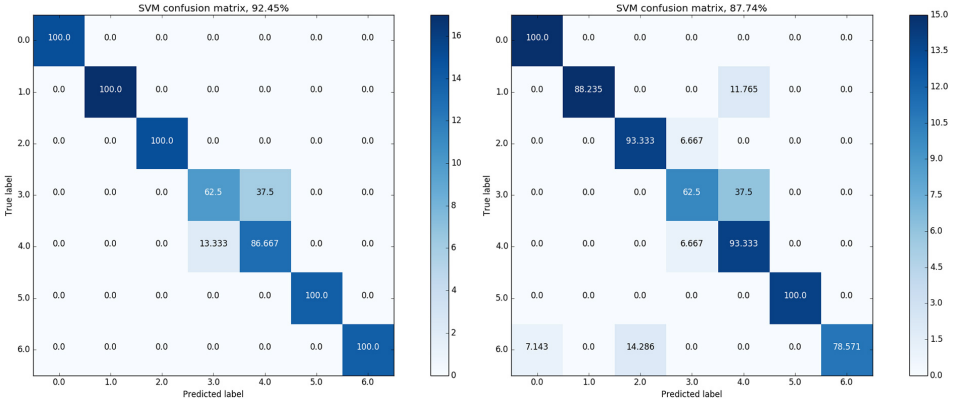
Im darauffolgenden Schritt wurden weitere Datensätze aus den nicht vorverarbeiteten Rohdaten generiert. Damit sollte überprüft werden, inwiefern sich die Erkennungsgenauigkeit im Vergleich zu den normierten Daten unterscheidet. Dabei lässt sich feststellen, dass die Erkennung zwar schlechter, aber dennnoch relativ gut ist. Man könnte jedoch vermuten, dass die Differenz bei zunehmender Anzahl an Personen und Gesten weiter ansteigt. Zudem scheint das neuronale Netz (siehe Tabelle 7.1) mit nur 0,58% Differenz weniger anfällig für die Varianz innerhalb der Rohdaten zu sein als die Support Vector Machine (siehe Tab. 7.2), bei der der Unterschied zu den normierten Daten bei knapp 6% liegt.

Als letzter Teil der Tests wurden Gesten aus dem normierten Datensatz entfernt. Damit sollte der Zusammenhang zwischen der Erkennungsgenauigkeit und der Anzahl von verfügbaren Daten untersucht werden. Zuerst



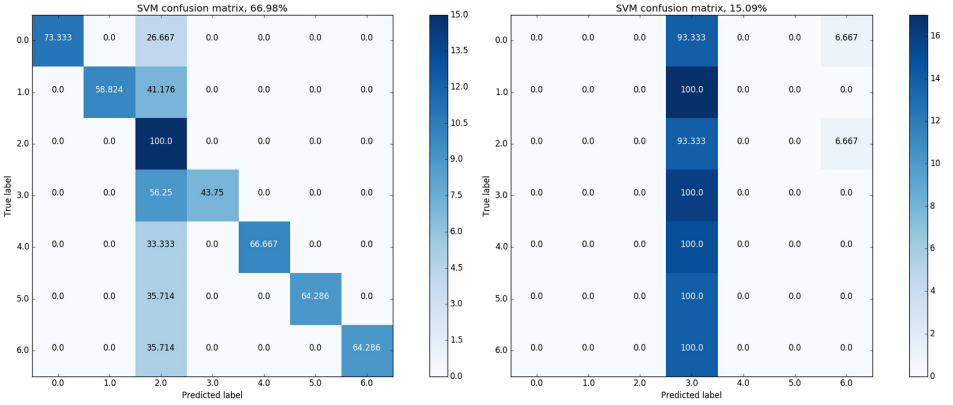
(a) Linear

(b) Polynom 2ten Grades



(c) Polynom 4ten Grades

(d) Polynom 6ten Grades



(e) Radiale Basisfunktion

(f) Sigmoid

Abbildung 7.6: Auswertung verschiedener Kernels. Der verwendete Kernel steht unter der jeweiligen Konfusionsmatrix.

Durchlauf	Normalisiert	Rohdaten	75% d. Daten	50% d. Daten
1	81,13%	82,08%	67,09%	66,04%
2	87,74%	81,13%	74,68%	71,70%
3	74,53%	77,36%	83,54%	75,47%
4	79,25%	84,91%	75,95%	67,92%
5	82,08%	78,30%	81,01%	81,13%
6	77,36%	73,58%	77,22%	69,81%
7	86,79%	85,71%	73,42%	73,58%
8	76,42%	80,19%	82,28%	79,25%
9	75,47%	79,25%	70,89%	64,15%
10	83,96%	76,42%	72,15%	77,36%
Durchschnitt	**80,47%**	**79,89%**	**75,82%**	**72,64%**

Tabelle 7.1: Ergebnisse verschiedener Datensätze, welche durch ein neuronales Netz, bestehend aus einem Hidden Layer und 450 Neuronen klassifiziert wurden. Die Verkleinerungen des Datensatzes, dargestellt in den letzten zwei Spalten, beziehen sich auf den normierten Datensatz.

wurde der Datensatz um ein Viertel, auf etwa 260 Daten, verkleinert. Das andere Mal wurde nur die Hälfte, also 175 Daten, verarbeitet. Es lässt sich beobachten, dass die Genauigkeit mit der Abnahme der Daten um bis zu 9% sinkt (siehe Tabelle 7.2). Im Umkehrschluss lässt sich annehmen, dass diese mit zunehmend mehr Daten steigt. Dieses Ergebnis deckt sich zudem mit der Lernkurve der Support Vector Machine in Abbildung 7.3.

7.5 Zusammenfassung

Zusammenfassend lässt sich sagen, dass die Support Vector Machine durchschnittlich besser abgeschnitten hat als das neuronale Netz. Nichtsdestotrotz konnten beide Algorithmen den normierten sowie den gesamten Datensatz am besten klassifizieren. Bei der Support Vector Machine ist die Wahl des Kernels entscheidend, wohingegen beim neuronalen Netz die Anzahl der Neuronen und Hidden Layer ausschlaggebend ist.

Eine andere Eigenschaft, die jedoch nur subjektiv gemessen wurde, ist die Dauer der Trainingsphase. Während die Support Vector Machine bereits nach wenigen Sekunden ein Ergebnis des zu klassifizierenden Testsets geliefert hat, konnte dieser Prozess beim neuronalen Netz bis zu mehreren Minuten dauern.

Zuletzt sollte gesagt werden, dass die vorgestellten Ergebnisse nicht verallgemeinert werden dürfen und sich lediglich auf die für diese Arbeit gesammelten Daten beziehen. Es ist durchaus möglich, dass die Algorithmen mit den präsentierten Einstellungen eine andere Problemstellung nicht zu-

Durchlauf	Normalisiert	Rohdaten	75% d. Daten	50% d. Daten
1	87,62%	76.42%	83,75%	77,36%
2	92,45%	84.91%	75,00%	84,91%
3	83,81%	82.08%	70,00%	75,47%
4	88,57%	83.02%	86,25%	79,25%
5	90,48%	78.30%	82,50%	83,02%
6	85,71%	83.96%	90,00%	71,70%
7	81,90%	79.25%	78,75%	69,81%
8	86,67%	87.74%	77,50%	73,58%
9	84,76%	77.34%	87,50%	86,79%
10	89,52%	81.13%	73,75%	81,13%
Durchschnitt	**87,15%**	**81.42%**	**80,50%**	**78,30%**

Tabelle 7.2: Auswertung verschiedener Datensätze mit einer Support Vector Machine mit einem Polynom zweiten Grades als Kernel. Die Verkleinerungen des Datensatzes, dargestellt in den letzten zwei Spalten, beziehen sich auf den normierten Datensatz.

friedenstellend lösen können. Deshalb sollte auch nicht behauptet werden, dass die Support Vector Machine grundsätzlich besser ist als das neuronale Netz. Es gilt immer die Aufgabenstellung zu beachten.

Kapitel 8

Zusammenfassung und Ausblick

8.1 Zusammenfassung

In dieser Arbeit wurden Auswertungen zweier maschineller Lernalgorithmen, der Support Vector Machine und dem neuronalen Netz, in Bezug auf die Erkennungsgenauigkeit von 3D-Gesten vorgestellt.

Dazu wurden zuerst die zwei Themenfelder Gestenerkennung und maschinelles Lernen, welche beide aktive Forschungsdisziplinen sind, betrachtet. Besonders durch den zunehmenden Fortschritt der Sensortechnik, allen voran die Entwicklung kostengünstiger Tiefensensoren, werden bestimmte Aufgaben der Gestenerkennung vereinfacht. Mit solch einem Tiefensensor, der Leap Motion, wurden für diese Arbeit sieben selbstdefinierte Gesten aufgezeichnet. Durch die anschließende Vorverarbeitung wurden die Handgröße sowie die exakte Position der Hand herausgerechnet und vereinheitlicht.

Daraufhin konnten die Algorithmen mit einem Teil der Daten trainiert werden, wodurch Zusammenhänge zwischen dem Merkmalsvektor und der jeweiligen Geste entstehen. Mit einem vorenthaltenen Teil der Daten wurden diese Zusammenhänge und somit die Erkennungsgenauigkeit getestet. Durch wiederholte, manuelle Anpassungen der jeweiligen Parameter konnten beide Algorithmen letztendlich eine Genauigkeit von über 93% erreichen. Allerdings hat sich herausgestellt, dass die Support Vector Machine mit durchschnittlich 87,15% um 7% besser als das neuronale Netz abschnitt. Außerdem wirkten sich die Normierung sowie die zunehmende Verfügbarkeit der Daten positiv auf die Erkennung aus. Auffallend bei allen Tests war, dass Gesten, welche sich stark ähneln, schwerer differenziert werden konnten.

Bei der Support Vector Machine war die Wahl des Kernels entscheidend, wobei ein Polynom zweiten Grades das beste Ergebnis erzielen konnte. Bei dem neuronalen Netz dagegen war vor allem die Struktur des Netzes ausschlaggebend, wobei sich ein Hidden Layer bestehend aus 450 Neuronen als

optimal herausgestellt hat.

Es soll auch noch mal an dieser Stelle erwähnt werden, dass diese Ergebnisse nicht verallgemeinert werden können und je nach Problemstellung differenziert werden muss. Lediglich der Prozess der Datenerhebung, die Auswahl der Merkmale, deren Vorverarbeitung, sowie die darauffolgenden Tranings- und Testphasen und die damit zusammenhängenden Anpassungen an den Parametern ist von Fall zu Fall ähnlich.

Mit Hilfe einer selbstentwickelten Software, wurde dieser gesamte Prozess und die Auswertungen realisiert. Für die Implementierung der maschinellen Lernalgorithmen wurde auf zwei Bibliotheken, TensorFlow und scikit-learn, zurückgegriffen.

8.2 Ausblick

Für diese Arbeit wurde mit 350 Daten ein relativ kleiner Datensatz verwendet. Für weitere Untersuchungen der Erkennungsgenauigkeit könnten mehr Daten, vor allem mit mehreren Personen, gesammelt werden.

Weitere Arbeit könnte zudem geleistet werden, indem überprüft wird, welche der 25 ausgewählten Merkmale wirklich relevant sind und ob die Anzahl nicht minimiert werden könnte. Dies würde den Merkmalsvektor verringern und dem Fluch der Dimensionalitäten entgegenwirken, wodurch anzunehmen ist, dass die Algorithmen die Gesten besser klassifizieren können. In dem Zusammenhang könnte auch überprüft werden, ob es sinnvoll ist andere Informationen, die die Leap Motion bereitstellt, zu verarbeiten. Beispielsweise könnte die Geschwindigkeit der einzelnen Finger als Merkmal betrachtet werden, womit sich anschließend die unterschiedlichen Geschwindigkeiten der Gesten herausrechnen lassen. Des Weiteren könnte man die Neigung der Hand zum Sensor normieren. Jedoch muss je nach Geste darauf geachtet werden, dass dabei keine wichtigen Eigenschaften zur Differenzierbarkeit verloren gehen. Mit der Normierung dieser zwei Eigenschaften, der Geschwindigkeit und der Neigung, wären alle personenspezifischen Informationen standardisiert, was eine höhere Erkennungsgenauigkeit zur Folge haben würde.

Abgesehen von der Untersuchung der Daten und Merkmale, könnten die vorliegenden Algorithmen detailliert untersucht und optimiert werden. Diese bieten schließlich weitaus mehr Parameter, die angepasst werden können, als nur den Kernel und die Netzstruktur.

Anschließend, losgelöst von der Untersuchung der Klassifizierungsgenauigkeit der Algorithmen, könnte das Live-Signal der Leap Motion verarbeitet werden, damit Gesten in Echtzeit erkannt werden können. In Folge dessen wäre es auch möglich nach Erkennen einer Geste eine bestimme Aktion auszuführen, um somit eine Software oder ähnliches steuern zu können.

Anhang A

Technische Informationen

Die Software wurde ausschließlich auf Mac OS X (10.11.6) entwickelt und getestet. Dennoch müsste durch die Verfügbarkeit der einzelnen Bibliotheken die Inbetriebnahme auf Windows und Linux ebenfalls möglich sein.

Im Folgenden werden die verwendeten Versionen der Abhängigkeiten aufgelistet, um eine fehlerfrei Inbetriebnahme zu garantieren.

- Python 2.7.12
- Leap Motion SDK 2.3.1
- PyOpenGL 3.1.0
- TensorFlow 0.11.0rc0
- scikit-learn 0.18.1
- Numpy 1.11.2
- Matplotlib 1.5.3

Wie zu erkennen ist, werden nicht immer die aktuellsten Versionen verwendet. Dies hängt damit zusammen, dass die neueste Leap Motion SDK (Version 3.2.0) nicht auf dem Mac unterstützt wird und die eingesetzte Version wiederum nicht mit Python 3 kompatibel ist. Des Weiteren wurde während des Verfassens dieser Arbeit TensorFlow 1.0 veröffentlicht, welche nun offiziell für den Produktiveinsatz vorgesehen ist (siehe Sandjideh 2017).

Falls eine eine andere Python Distribution, beispielsweise durch Homebrew installiert wurde, muss dieser alternative Pfad in der Leap Motion Bibliothek referenziert werden. Siehe dazu Leap Motion (2016c).

Anhang B

Inhalt der CD-ROM

Pfad: /

Bilder/	In der Arbeit verwendete Bilder
leapGesture/	Entwickelte Software
Poster.jpg	Tag der Medien Poster in geringerer Auflösung
Thesis.pdf	Vorliegende Bachelorarbeit

Pfad: leapGesture/

data/	Aufgezeichnete Daten, sowie der verwendeten Trainings- und Testdatensatz
libs/	Leap Motion SDK
src/	Sämtlicher entwickelter Sourcecode
tmp/	Trainierte Modelle Neuronaler Netze

Pfad: leapGesture/src/

leapController.py	Modul zur Aufzeichnung und Speicherung der Daten
neuralNet.py	Implementierung eines Neuronalen Netzes
normalize.py	Modul zur Normalisierung der Daten
plotMatrix.py	Ausgabe der Confusionmatrix
processFiles.py	Generierung des Training- und Testsets
ringBuffer.py	Zwischenspeicher für die empfangenen Daten per Leap Motion
run.py	Modul zur Ausführung der Tests
svm.py	Implementierung einer Support Vector Machine

Quellenverzeichnis

Literatur

Abadi, Martin u. a. (2015). *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. URL: http://tensorflow.org/ (siehe S. 19).

Binh, Nguyen D., Enokida Shuichi und Toshiaki Ejima (2005). „Real-Time Hand Tracking and Gesture Recognition System". In: *ICGST Conference on Graphics, Vision and Image Processing, GVIP-05*. Kairo (siehe S. 5).

Bishop, Christopher M. (1995). *Neural Networks for Pattern Recognition*. New York: Oxford University Press (siehe S. 15).

— (2006). *Pattern Recognition and Machine Learning*. New York: Springer (siehe S. 9, 10, 15).

Burges, Christopher J. C. (1998). „A Tutorial on Support Vector Machines for Pattern Recognition". *Data Mining and Knowledge Discovery* 2.2, S. 121–167 (siehe S. 16).

Cortes, Corinna und Vladimir Vapnik (1995). „Support-Vector Networks". *Machine Learning* 20.3, S. 273–297 (siehe S. 16, 17).

Cristianini, Nello und John Shawe-Taylor (2000). *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods*. New York: Cambridge University Press (siehe S. 16).

Domingos, Pedro (2012). „A Few Useful Things to Know About Machine Learning". *Communications of the ACM* 55.10, S. 78–87 (siehe S. 13).

D'Orazio, Tiziana u. a. (2016). „Recent trends in gesture recognition: how depth data has improved classical approaches". *Image and Vision Computing* 52, S. 56–72 (siehe S. 2, 4).

Duda, Richard O., Peter E. Hart und David G. Stork (2000). *Pattern Classification*. 2nd Edition. Wiley (siehe S. 13–15).

Garg, Pragati, Naveen Aggarwal und Sanjeev Sofat (2009). „Vision Based Hand Gesture Recognition". *International Journal of Computer, Electrical, Automation, Control and Information Engineering* 3.1, S. 186–191 (siehe S. 1).

Han, Jungong u. a. (2013). „Enhanced Computer Vision with Microsoft Kinect Sensor: A Review“. *IEEE Transactions on Cybernetics* 43.5 (siehe S. 1, 4).

Huang, Thomas S. und Vladimir I. Pavlovic (1995). „Hand Gesture Modeling, Analysis, and Synthesis“. In: *Proc. of IEEE International Workshop on Automatic Face and Gesture Recognition*, S. 73–79 (siehe S. 10).

Kotsiantis, Sotiris B. (2007). „Supervised Machine Learning: A Review of Classification Techniques“. *Informatica* 31.3, S. 249–268 (siehe S. 14–16).

Kurakin, Alexey, Zhengyou Zhang und Z. Liu (2012). „A real time system for dynamic hand gesture recognition with a depth sensor“. In: *2012 Proceedings of the 20th European Signal Processing Conference (EUSIPCO)*, S. 1975–1979 (siehe S. 5).

LaViola Jr., Joseph J. (2013). „3D Gestural Interaction: The State of the Field“. *ISRN Artificial Intelligence* 2013 (siehe S. 5).

Lu, Wei, Zheng Tong und Jinghui Chu (2016). „Dynamic Hand Gesture Recognition With Leap Motion Controller“. *IEEE Signal Processing Letters* 23.9, S. 1188–1192 (siehe S. 5).

Marin, Giulio, Fabio Dominio und Pietro Zanuttigh (2014). „Hand gesture recognition with leap motion and kinect devices“. In: *2014 IEEE International Conference on Image Processing (ICIP)* (siehe S. 5).

Mitchell, Tom M. (1997). *Machine learning*. Internat. ed. New York: McGraw-Hill (siehe S. 2, 5, 14, 15).

Mitra, Sushmita und Tinku Acharya (2007). „Gesture Recognition: A Survey“. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 37.3, S. 311–324 (siehe S. 1).

Murakami, Kouichi und Hitomi Taguchi (1991). „Gesture Recognition using Recurrent Neural Networks“. In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, S. 237–242 (siehe S. 5).

Pavlovic, Vladimir I., Rajeev Sharma und Thomas S. Huang (1997). „Visual Interpretation of Hand Gestures for Human-Computer Interaction: A Review“. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 19.7, S. 677–695 (siehe S. 4).

Pedregosa, Fabian u. a. (2011). „Scikit-learn: Machine Learning in Python“. *Journal of Machine Learning Research* 12, S. 2825–2830 (siehe S. 19).

Premaratne, Prashan (2014). *Human Computer Interaction Using Hand Gestures*. Cognitive Science and Technology. Singapur: Springer (siehe S. 4).

Rautaray, Siddharth S. und Anupam Agrawal (2015). „Vision based hand gesture recognition for human computer interaction: a survey“. *Artificial Intelligence Review* 43.1, S. 1–54 (siehe S. 5).

Wachsmuth, Ipke und Martin Fröhlich (1998). *Gesture and Sign Language in Human-Computer Interaction*. Bd. 1371. Lecture Notes in Computer Science. Berlin: Springer (siehe S. 1).

Weichert, Frank u. a. (2013). „Analysis of the Accuracy and Robustness of the Leap Motion Controller“. *Sensors* 13.5, S. 6380–6393 (siehe S. 5, 7).
Wölfel, Matthias (2013). „3D Gestenerkennung als moderne Mensch-Maschine-Schnittstelle“. *uDay* XI (siehe S. 1).

Online-Quellen

Amazon (2013). *Leap Motion Controller für Mac/PC*. URL: https://images-na.ssl-images-amazon.com/images/I/61frayN%2Bg7L._SL1500_.jpg (besucht am 03. 02. 2017) (siehe S. 8).
Colgan, Alex (2014). *How Does the Leap Motion Controller Work?* URL: http://blog.leapmotion.com/hardware-to-software-how-does-the-leap-motion-controller-work/ (besucht am 02. 02. 2017) (siehe S. 7).
karenu (2012). *machine learning - What is the relation between the number of Support Vectors and training data and classifiers performance? - Stack Overflow*. URL: https://i.stack.imgur.com/1gvce.png (besucht am 09. 02. 2017) (siehe S. 17).
Leap Motion (2016a). *API Overview - Leap Motion Python SDK v3.2 Documentation*. URL: https://developer.leapmotion.com/documentation/python/devguide/Leap_Overview.html (besucht am 05. 02. 2017) (siehe S. 8, 10).
— (2016b). *Pointables - Leap Motion Python SDK v3.2 Documentation*. URL: https://developer.leapmotion.com/documentation/python/api/Leap.Pointable.html (besucht am 05. 02. 2017) (siehe S. 8).
— (2016c). *Setting Up a Project - Leap Motion Python SDK v2.3*. URL: https://developer.leapmotion.com/documentation/v2/python/devguide/Project_Setup.html#using-a-different-python-distribution (besucht am 11. 02. 2017) (siehe S. 35).
— (2017). *Leap Motion for Mac and PC*. URL: https://www.leapmotion.com/product/desktop (besucht am 02. 02. 2017) (siehe S. 8).
Sandjideh, Amy McDonald (2017). *Announcing TensorFlow 1.0*. URL: https://developers.googleblog.com/2017/02/announcing-tensorflow-10.html (besucht am 19. 02. 2017) (siehe S. 35).
scikit-learn developers (2016a). *Confusion matrix – scikit-learn 0.18.1 documentation*. URL: http://scikit-learn.org/stable/auto_examples/model_selection/plot_confusion_matrix.html (besucht am 17. 02. 2017) (siehe S. 22).
— (2016b). *Plotting Learning Curves – scikit-learn 0.18.1 documentation*. URL: http://scikit-learn.org/stable/auto_examples/model_selection/plot_learning_curve.html (besucht am 17. 02. 2017) (siehe S. 22).
Smilkov, Daniel und Shan Carter (2016). *A Neural Network Playground*. URL: http://playground.tensorflow.org/ (besucht am 03. 02. 2017) (siehe S. 16).

SparkFun (2013). *Leap Motion Teardown*. URL: https://cdn.sparkfun.com/r/600-600/assets/b/6/6/d/5/51c471f2ce395ffa74000000.jpg (besucht am 02. 02. 2017) (siehe S. 8).